Program-02: Simulate the following CPU scheduling algorithms to find turnaround time and waiting time:
a) FCFS, b) SJF, c) Round Robin and d) Priority.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int id;
    int burst_time;
    int priority;
} Process;

void fcfs_scheduling(int n, int burst_times[]) {
    int waiting_time[n], turnaround_time[n];
    waiting_time[0] = 0;
    turnaround_time[0] = burst_times[0];

    for (int i = 1; i < n; i++) {
        waiting_time[i] = waiting_time[i - 1] + burst_times[i - 1];
        turnaround_time[i] = waiting_time[i] + burst_times[i];
    }

    printf("FCFS Scheduling\n");
    printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst_times[i], waiting_time[i], turnaround_time[i]);
    }
}

int compare_sjf(const void *a, const void *b) {
    return ((Process *)a)->burst_time - ((Process *)b)->burst_time;
}

void sjf_scheduling(int n, Process processes[]) {
    int waiting_time[n], turnaround_time[n];

    qsort(processes, n, sizeof(Process), compare_sjf);

    waiting_time[0] = 0;
    turnaround_time[0] = processes[0].burst_time;

    for (int i = 1; i < n; i++) {
        waiting_time[i] = waiting_time[i - 1] + processes[i - 1].burst_time;
        turnaround_time[i] = waiting_time[i] + processes[i].burst_time;
    }

    printf("SJF Scheduling\n");
    printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst_time, waiting_time[i],
turnaround_time[i]);
```

```
    }
}

void round_robin_scheduling(int n, int burst_times[], int quantum) {
    int remaining_times[n], waiting_time[n], turnaround_time[n];
    int t = 0;

    for (int i = 0; i < n; i++) {
        remaining_times[i] = burst_times[i];
    }

    while (1) {
        int done = 1;
        for (int i = 0; i < n; i++) {
            if (remaining_times[i] > 0) {
                done = 0;
                if (remaining_times[i] > quantum) {
                    t += quantum;
                    remaining_times[i] -= quantum;
                } else {
                    t += remaining_times[i];
                    waiting_time[i] = t - burst_times[i];
                    remaining_times[i] = 0;
                }
            }
        }
        if (done) {
            break;
        }
    }

    for (int i = 0; i < n; i++) {
        turnaround_time[i] = burst_times[i] + waiting_time[i];
    }

    printf("Round Robin Scheduling\n");
    printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst_times[i], waiting_time[i], turnaround_time[i]);
    }
}

int compare_priority(const void *a, const void *b) {
    return ((Process *)a)->priority - ((Process *)b)->priority;
}

void priority_scheduling(int n, Process processes[]) {
    int waiting_time[n], turnaround_time[n];

    qsort(processes, n, sizeof(Process), compare_priority);

    waiting_time[0] = 0;
```

```
    turnaround_time[0] = processes[0].burst_time;

    for (int i = 1; i < n; i++) {
        waiting_time[i] = waiting_time[i - 1] + processes[i - 1].burst_time;
        turnaround_time[i] = waiting_time[i] + processes[i].burst_time;
    }

    printf("Priority Scheduling\n");
    printf("Process ID\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burst_time,
processes[i].priority, waiting_time[i], turnaround_time[i]);
    }
}

int main() {
    int n, quantum;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    int burst_times[n];
    Process processes[n];

    printf("Enter burst times for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Burst Time for P%d: ", i + 1);
        scanf("%d", &burst_times[i]);
        processes[i].id = i + 1;
        processes[i].burst_time = burst_times[i];
    }

    printf("Enter the quantum time for Round Robin (0 to skip): ");
    scanf("%d", &quantum);

    if (quantum > 0) {
        round_robin_scheduling(n, burst_times, quantum);
    }

    printf("Enter priorities for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Priority for P%d: ", i + 1);
        scanf("%d", &processes[i].priority);
    }

    fcfs_scheduling(n, burst_times);
    sjf_scheduling(n, processes);
    priority_scheduling(n, processes);

    return 0;
}
```

OUTPUT:

Enter the number of processes: 4
Enter burst times for each process:
Burst Time for P1: 10
Burst Time for P2: 5
Burst Time for P3: 8
Burst Time for P4: 12
Enter the quantum time for Round Robin (0 to skip): 4

Round Robin Scheduling

Process ID	Burst Time	Waiting Time	Turnaround Time
P1	10	21	31
P2	5	16	21
P3	8	17	25
P4	12	23	35

Enter priorities for each process:
Priority for P1: 2
Priority for P2: 1
Priority for P3: 4
Priority for P4: 3

FCFS Scheduling

Process ID	Burst Time	Waiting Time	Turnaround Time
P1	10	0	10
P2	5	10	15
P3	8	15	23
P4	12	23	35

SJF Scheduling

Process ID	Burst Time	Waiting Time	Turnaround Time
P2	5	0	5
P3	8	5	13
P1	10	13	23
P4	12	23	35

Priority Scheduling

Process ID	Burst Time	Priority	Waiting Time	Turnaround Time
P2	5	1	0	5
P1	10	2	5	15
P4	12	3	15	27
P3	8	4	27	35